```
/*
Package main is the entry point for the Car Garage Management application.
It uses the Gin web framework for routing and MongoDB as the database.
*/

package main

import (
	"context"
	"fmt"
	"log"
	"net/http"

	"github.com/gin-gonic/gin"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

// Car represents the structure of a car in the system.
type Car struct {
	ID     string `json:"car_id" bson:"_id,omitempty"`
	Brand  string `json:"brand" bson:"brand"`
	Model  string `json:"model" bson:"model"`
	Status string `json:"status" bson:"status"`
}

var collection *mongo.Collection

// init initializes the MongoDB connection and sets up the collection.
```

```
func init() {
	clientOptions := options.Client().ApplyURI("mongodb+srv://notexist123:notexist123@cluster0.v1k4aao.mongodb.net/garage")
	client, err := mongo.Connect(context.Background(), clientOptions)
	if err != nil {
		log.Fatal(err)
	}

	if err = client.Ping(context.Background(), nil); err != nil {
		log.Fatal(err)
	}

	collection = client.Database("garage").Collection("cars")
}

// main is the entry point of the application.
func main() {
	router := gin.Default()

	// HTML endpoint to render the main page
	router.GET("/", func(c *gin.Context) {
		c.HTML(http.StatusOK, "index.html", nil)
	})

	// API group for car-related routes
	apiGroup := router.Group("/api/cars")
	{
		apiGroup.POST("/", addCar)
```

```
		apiGroup.GET("/", getCars)
		apiGroup.PUT("/:car_id", updateCarStatus)
		apiGroup.DELETE("/:car_id", deleteCar)
		apiGroup.DELETE("/", deleteAllCars)
	}

	// Run the application on port 5000
	if err := router.Run(":5000"); err != nil {
		log.Fatal(err)
	}
}

// addCar handles the addition of a new car to the system.
func addCar(c *gin.Context) {
	var car Car
	if err := c.ShouldBindJSON(&car); err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request payload"})
		return
	}

	if car.Brand == "" || car.Model == "" {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Brand and Model are required"})
		return
	}

	if findCarByBrandAndModel(car.Brand, car.Model) != nil {
		c.JSON(http.StatusConflict, gin.H{"error": "Car already exists"})
		return
	}
```

```
	car.Status = "In Garage"
	result, err := collection.InsertOne(context.Background(), car)
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}

	c.JSON(http.StatusCreated, gin.H{"message": "Car added successfully", "car_id": result.InsertedID})
}

// getCars retrieves the list of cars in the garage.
func getCars(c *gin.Context) {
	var cars []Car
	cur, err := collection.Find(context.Background(), nil)
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}
	defer cur.Close(context.Background())

	for cur.Next(context.Background()) {
		var car Car
		err := cur.Decode(&car)
		if err != nil {
			c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
			return
		}
```

```
		cars = append(cars, car)
	}

	c.JSON(http.StatusOK, gin.H{"cars": cars})
}

// updateCarStatus updates the status of a specific car.
func updateCarStatus(c *gin.Context) {
	carID := c.Param("car_id")
	var data map[string]string
	if err := c.ShouldBindJSON(&data); err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request payload"})
		return
	}

	newStatus, exists := data["status"]
	if !exists || (newStatus != "In Garage" && newStatus != "Under Repair" && newStatus != "Completed") {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid status"})
		return
	}

	result, err := collection.UpdateOne(context.Background(), bson.M{"_id": carID}, bson.M{"$set": bson.M{"status": newStatus}})
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}
```

```
	if result.MatchedCount == 0 {
		c.JSON(http.StatusNotFound, gin.H{"error": "Car not found"})
		return
	}

	c.JSON(http.StatusOK, gin.H{"message": "Car status updated successfully"})
}

// deleteCar deletes a specific car from the system.
func deleteCar(c *gin.Context) {
	carID := c.Param("car_id")
	result, err := collection.DeleteOne(context.Background(), bson.M{"_id": carID})
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}

	if result.DeletedCount == 0 {
		c.JSON(http.StatusNotFound, gin.H{"error": "Car not found"})
		return
	}

	c.JSON(http.StatusOK, gin.H{"message": "Car deleted successfully"})
}

// deleteAllCars deletes all cars from the system.
func deleteAllCars(c *gin.Context) {
	result, err := collection.DeleteMany(context.Background(), bson.M{})
	if err != nil {
```

```
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}

	c.JSON(http.StatusOK, gin.H{"message": fmt.Sprintf("%d cars deleted successfully", result.DeletedCount)})
}

// findCarByBrandAndModel searches for a car in the database by brand and model.
func findCarByBrandAndModel(brand, model string) *Car {
	var car Car
	err := collection.FindOne(context.Background(), bson.M{"brand": brand, "model": model}).Decode(&car)
	if err != nil {
		return nil
	}
	return &car
}
```

Certainly! Here's an extended documentation with instructions on how to run the provided Golang code:

go

Copy code

```
/*
Package main is the entry point for the Car Garage Management application.
It uses the Gin web framework for routing and MongoDB as the database.
*/

package main
```

```
import (
	"context"
	"fmt"
	"log"
	"net/http"

	"github.com/gin-gonic/gin"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

// Car represents the structure of a car in the system.
type Car struct {
	ID     string `json:"car_id" bson:"_id,omitempty"`
	Brand  string `json:"brand" bson:"brand"`
	Model  string `json:"model" bson:"model"`
	Status string `json:"status" bson:"status"`
}

var collection *mongo.Collection

// init initializes the MongoDB connection and sets up the collection.
func init() {
	clientOptions := options.Client().ApplyURI("mongodb+srv://notexist123:notexist123@cluster0.v1k4aao.mongodb.net/garage")
	client, err := mongo.Connect(context.Background(), clientOptions)
```

```
	if err != nil {
		log.Fatal(err)
	}

	if err = client.Ping(context.Background(), nil); err != nil {
		log.Fatal(err)
	}

	collection = client.Database("garage").Collection("cars")
}

// main is the entry point of the application.
func main() {
	router := gin.Default()

	// HTML endpoint to render the main page
	router.GET("/", func(c *gin.Context) {
		c.HTML(http.StatusOK, "index.html", nil)
	})

	// API group for car-related routes
	apiGroup := router.Group("/api/cars")
	{
		apiGroup.POST("/", addCar)
		apiGroup.GET("/", getCars)
		apiGroup.PUT("/:car_id", updateCarStatus)
		apiGroup.DELETE("/:car_id", deleteCar)
		apiGroup.DELETE("/", deleteAllCars)
	}
```

```
	// Run the application on port 5000
	if err := router.Run(":5000"); err != nil {
		log.Fatal(err)
	}
}

// addCar handles the addition of a new car to the system.
func addCar(c *gin.Context) {
	var car Car
	if err := c.ShouldBindJSON(&car); err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request payload"})
		return
	}

	if car.Brand == "" || car.Model == "" {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Brand and Model are required"})
		return
	}

	if findCarByBrandAndModel(car.Brand, car.Model) != nil {
		c.JSON(http.StatusConflict, gin.H{"error": "Car already exists"})
		return
	}

	car.Status = "In Garage"
	result, err := collection.InsertOne(context.Background(), car)
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
```

```
		return
	}

	c.JSON(http.StatusCreated, gin.H{"message": "Car added successfully", "car_id": result.InsertedID})
}

// getCars retrieves the list of cars in the garage.
func getCars(c *gin.Context) {
	var cars []Car
	cur, err := collection.Find(context.Background(), nil)
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}
	defer cur.Close(context.Background())

	for cur.Next(context.Background()) {
		var car Car
		err := cur.Decode(&car)
		if err != nil {
			c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
			return
		}
		cars = append(cars, car)
	}

	c.JSON(http.StatusOK, gin.H{"cars": cars})
}
```

```
// updateCarStatus updates the status of a specific car.
func updateCarStatus(c *gin.Context) {
	carID := c.Param("car_id")
	var data map[string]string
	if err := c.ShouldBindJSON(&data); err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request payload"})
		return
	}

	newStatus, exists := data["status"]
	if !exists || (newStatus != "In Garage" && newStatus != "Under Repair" && newStatus != "Completed") {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid status"})
		return
	}

	result, err := collection.UpdateOne(context.Background(), bson.M{"_id": carID}, bson.M{"$set": bson.M{"status": newStatus}})
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}

	if result.MatchedCount == 0 {
		c.JSON(http.StatusNotFound, gin.H{"error": "Car not found"})
		return
	}
```

```
	c.JSON(http.StatusOK, gin.H{"message": "Car status updated successfully"})
}

// deleteCar deletes a specific car from the system.
func deleteCar(c *gin.Context) {
	carID := c.Param("car_id")
	result, err := collection.DeleteOne(context.Background(), bson.M{"_id": carID})
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}

	if result.DeletedCount == 0 {
		c.JSON(http.StatusNotFound, gin.H{"error": "Car not found"})
		return
	}

	c.JSON(http.StatusOK, gin.H{"message": "Car deleted successfully"})
}

// deleteAllCars deletes all cars from the system.
func deleteAllCars(c *gin.Context) {
	result, err := collection.DeleteMany(context.Background(), bson.M{})
	if err != nil {
		c.JSON(http.StatusInternalServerError, gin.H{"error": "Internal Server Error"})
		return
	}
```

```
	c.JSON(http.StatusOK, gin.H{"message": fmt.Sprintf("%d cars deleted successfully", result.DeletedCount)})
}

// findCarByBrandAndModel searches for a car in the database by brand and model.
func findCarByBrandAndModel(brand, model string) *Car {
	var car Car
	err := collection.FindOne(context.Background(), bson.M{"brand": brand, "model": model}).Decode(&car)
	if err != nil {
		return nil
	}
	return &car
}
```

How to Run:

Install Go Dependencies:

Open a terminal and navigate to the directory containing the main.go file. Run the following commands to download and install the required dependencies:

```
go get -u github.com/gin-gonic/gin
go get -u go.mongodb.org/mongo-driver/mongo
```

Run the Application:

After installing dependencies, run the following command in the terminal to start the application:

```
go run main.go
```

Access the Application:

Open your web browser and visit http://localhost:5000 to interact with the web application.